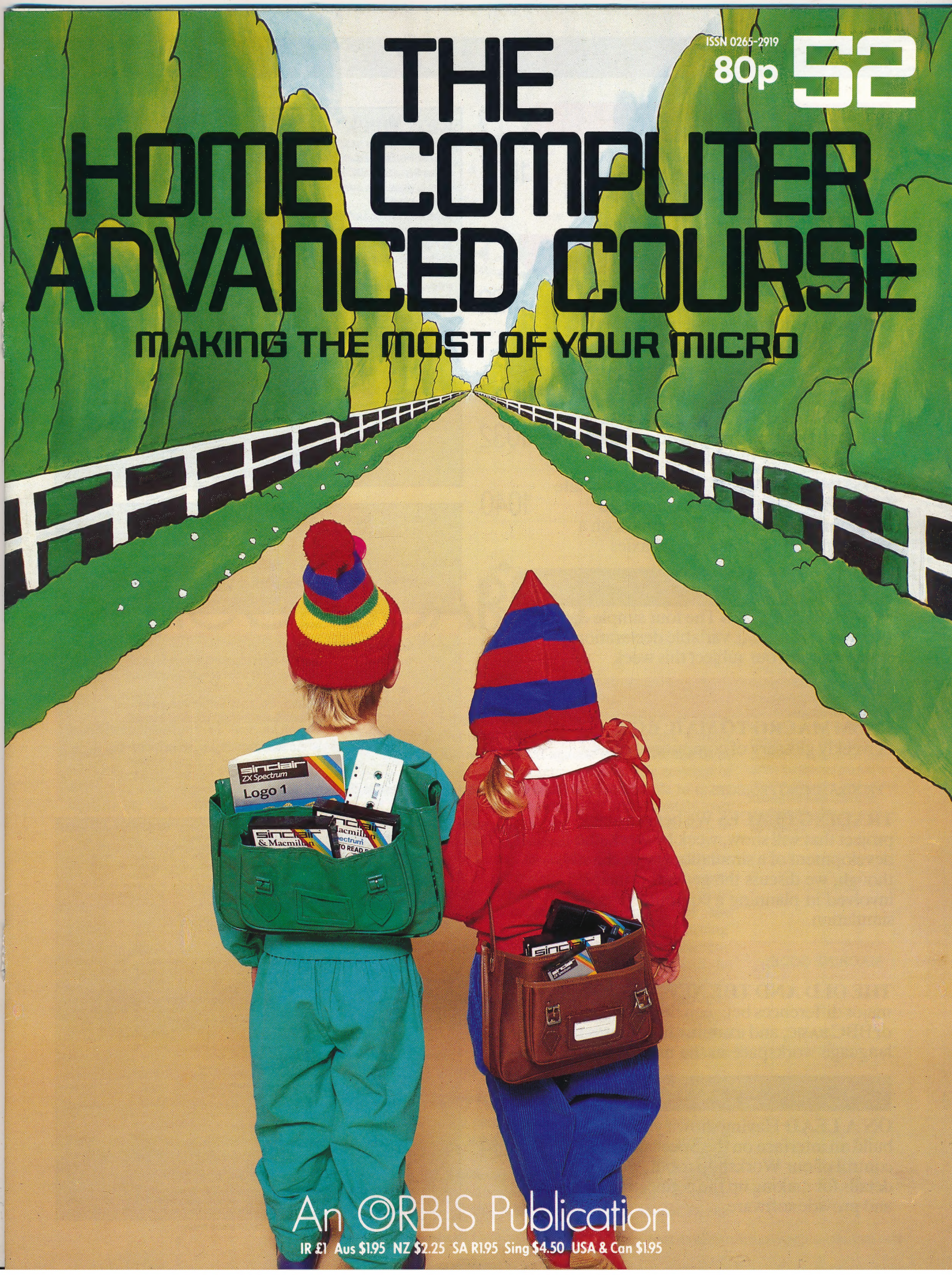
ISSN 0265-2919

80p 52

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



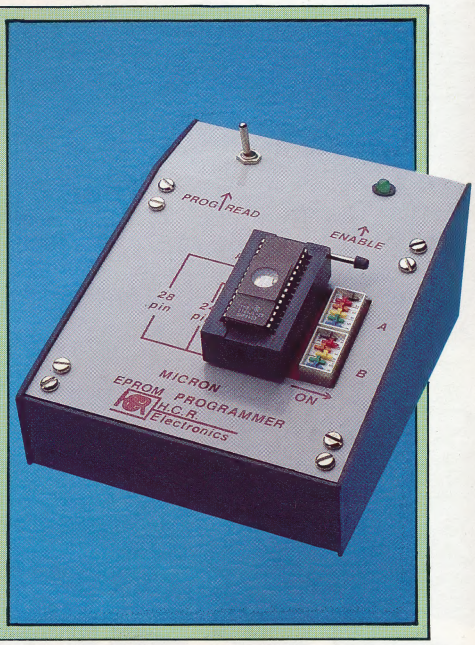
An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **We take a look at an EPROM programmer, an inexpensive device that allows BBC Micro users to place their own programs in ROM.**
- **The AMX mouse enables Macintosh-type applications on the BBC Micro. We examine how this package measures up.**
- **Using our robot's light sensors, we produce a 'line follower' program.**
- **We present the first module in our commercial trading game.**



Owing to the constantly rising costs of paper, ink, printing and transport, it is with considerable regret that THE HOME COMPUTER ADVANCED COURSE is forced to announce a price increase: UK 90p; IR£1.15. Readers who have taken out subscriptions will not have to pay the increase until the subscription is renewed.

Answers To Last Week's Quiz

1) Because the keyboard and mouse transmit infra-red signals, these may interfere with each other when several Portables are used in close proximity.

2) To run a new PASCAL program, first the program must be Edited (typed in), then Compiled into object code and finally it can be Run.

3) The Calendar option in the Microsoft Project allows the user to enter non-working days, such as Sundays and public holidays.

4) The most widely used application of the magnetic card is bank or credit cards.

Managing Editor Mike Wesley; **Editor** Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Technical Editor** Steve Colwill; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Sub Editor** Jonathan Kaye; **Contributors** Geoff Bains, Nick Walsh, Joe Pritchard, Steve Malone, Karl Dallas, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Manager** Peter Taylor-Medhurst; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

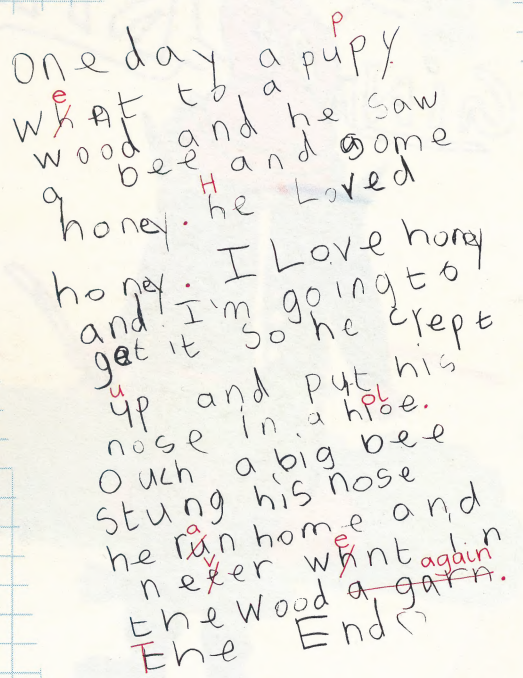
HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 80p/IR£1. Subscription: 6 months: £23.92. 1 Year: £47.84. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 80p. Subscription: 6 months air: £43.68. Surface: £34.84. 1 year air: £87.36. Surface: £69.68. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 80p. Subscription: 6 months air: £45.76. Surface: £34.84. 1 year air: £91.52. Surface: £69.68. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/80p. Subscription: 6 months air: £54.08. Surface: £34.84. 1 year air: £108.16. Surface: £69.68. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R1.95. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 80p. Subscription: 6 months air: £58.24. Surface: £34.84. 1 year air £116.48. Surface: £69.68. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: Aus$1.95. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZ$2.25. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.

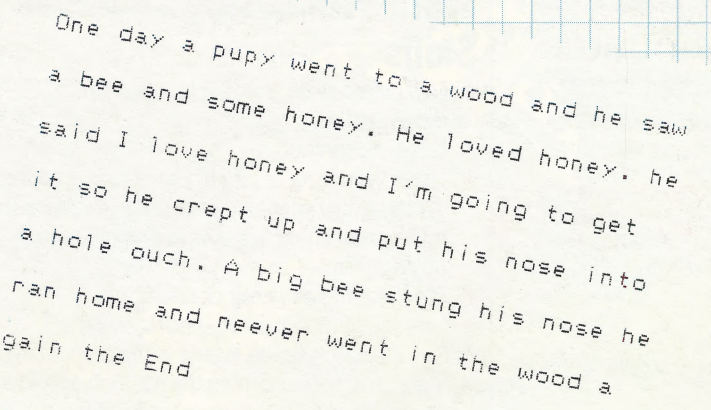
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.

NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH SET BY SIMON BRADY
DESIGN COURTESY OF START-RITE SHOES


One day a pupy
went to a
wood and he saw
bee and some
honey. he Loved
honey. I Love honey
and I'm going to
get it So he crept
up and put his
nose in a hole.
Ouch a big bee
stung his nose
he ran home and
never went again.
The Wood
The End


One day a pupy went to a wood and he saw a bee and some honey. He loved honey. he said I love honey and I'm going to get it so he crept up and put his nose into a hole ouch. A big bee stung his nose he ran home and neever went in the wood a gain the End

SCHOOL BOARDS

In this instalment of our education series, we look at the use of micros in helping to develop writing and drawing skills in young children. This application of computers is a significant and controversial development. Both skills embody strong creative elements and the introduction of computers into this area tends to be viewed with suspicion.

As a general rule, technical innovations meet with a similar response to political coups — people are often reluctant to experiment and will refuse to support new ideas until they have proved themselves. One of the most basic tools of education today, the exercise book, represented a revolutionary development when it first began to replace the slate. Previously, paper had mostly been used for recording business transactions, and for important work in universities, government, and the church. An expensive commodity, it was seen as quite unsuitable for children, who probably either wouldn't know how to use it, or would do so wastefully.

However, the introduction of exercise books gave students a permanent record of their work. It enabled them to divide their attentions between different written tasks without erasing earlier efforts. Although an expensive innovation, it was an immensely valuable one. However, as far as arguments of economy are concerned, similar objections are raised against the use of micros as were used against exercise books. The concept of one computer per child is unacceptable to many on grounds of cost alone and, unfortunately, cost has always been an overriding factor in determining educational policy.

Where micros are concerned there is also the difficulty of persuading teachers to accept a new technology that has already acquired a dubious reputation, principally as a result of the inheritance of the 'programmed learning' techniques discussed in the previous instalment. A close look at the way computers can be used in the classroom will dispel many objections and show that the computer not only increases efficiency — thus making it more economical than its high initial cost would suggest — but also has a valuable contribution to make to the creative processes of learning.

CREATIVE WRITING

Let's first consider the use of micros in helping to develop writing skills. Writing a story involves several different creative processes. After conceptualisation, a story needs much rewriting, editing and improving before it is complete. In this sense, writing is seen as being very similar to, say,

Word Play

One of the major problems children face when they attempt to write creatively is the inability of their handwriting skills to keep pace with their thoughts. In the process of forming letters correctly, it is all too easy to lose a train of thought. It is a characteristic of children's writing to appear disjointed for this reason. The word processor by itself does not solve this problem: it can take as long to find a key as it does to write the letter by hand. But with proper typing skills, which require fewer fine motor skills than handwriting, a child can overcome this difficulty to a certain extent. Since editing can be done on screen or on paper before the finished product is required, less time can be spent making the story appear on paper, so thoughts and words are more closely linked

Micro-Cognition
The use of a computer for word processing and drawing develops many of the same skills as writing and drawing by hand. But the computer, used properly, can actually teach or strengthen many skills better than learning by hand, and develops some additional skills as well. The tables list some of the cognitive skills computers teach or refine

Word Processing Cognitive Skills

- Fine motor skills (small finger and hand movements)
- Letter recognition
- Letter formation (a proper character set on the screen and printer can teach the correct way to make a letter, and it is always done the same way)
- Proofreading and error checking
- Spelling (a multitude of specific skills)
- Sequencing (placing events in the proper order)
- Visual memory (learning the position of the keys so you don't have to look at the keyboard)

Graphics Cognitive Skills

- Eye-hand co-ordination
- Fine motor skills
- Patterning
- Shape discrimination
- Cause-and-effect relationship
- Fluency (generating lots of ideas very quickly)
- Flexibility (lateral thinking: seeing old things in a new light)
- Imagination

painting and sculpture.

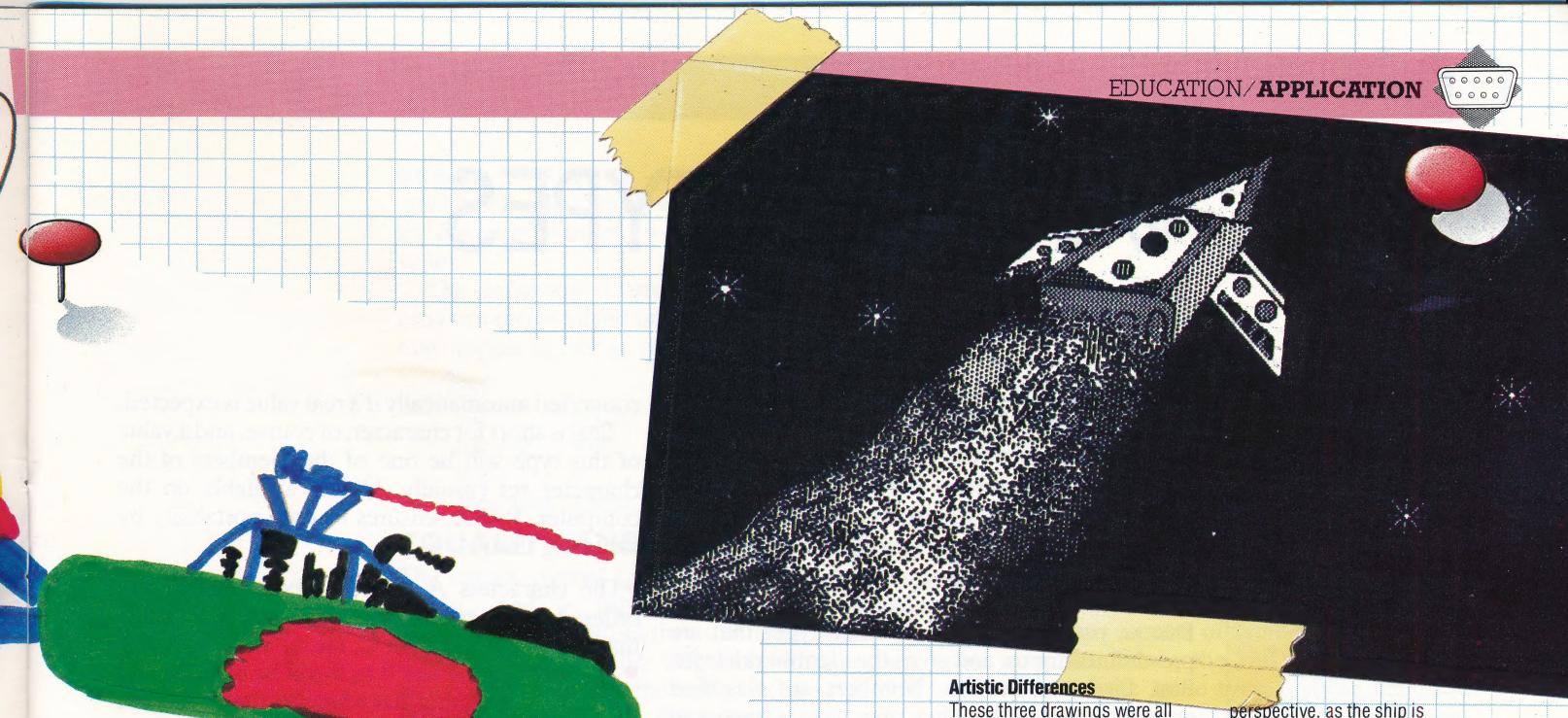
Unfortunately, the way many children are expected to produce written work in schools runs counter to the creative process. Frequently, the prime requirement of the teacher is that they should finish the work as quickly as possible. They must also present work tidily, with no crossing out or alteration. Editing, redrafting, and hence serious creative writing are often discouraged by the teacher's desire for neat work. The teacher's attitude is shaped largely by the need to process written work as quickly as possible, and untidy work takes longer to mark.

Allowing a child to use a computer as a word processor reconciles the 'tidy page' syndrome with serious writing skills. Children learn to edit, correct and improve their work and still finish with a neatly printed final copy. Text can be stored on disk and when the task is finished the child can hand the disk to the teacher, who can put it into a computer at home or in school and review it when convenient. If a computer network is used, the teacher can call up the child's file directly and check through the work. The end product is a neat copy on paper that even the most untidy writer can be proud of. There are also programs that run through a file to check spelling, pointing out and correcting any mistakes.

One possible drawback to this process is that the child may not — on seeing the corrected text — be made aware of his or her mistakes as would be the case with a marked exercise book. Many people who use word processors prefer to correct their manuscript on paper rather than on the screen. It is easier on the eyes, the paper is more portable than the computer, and it is possible to scribble comments over the text. Unless a computer is developed that is as user-friendly as a piece of paper for correcting written work, it is likely that pencil and paper will be with us for many years yet.

A child learning to use a computer elicits much comment about the 'barrier of the keyboard', yet little is heard about the 'barrier of the pencil' in learning to write by hand. One of the main objectives of teachers of infants or primary school pupils is to teach the children to write. Initially they must learn to hold the pencil correctly, then learn the shapes of the letters and, finally, master the skill of reproducing those shapes on paper. Letters must be drawn in a particular way and parts of letters in a particular order. The child must make words a uniform size and ensure they are laid down in a straight line. These skills take years of hard work to master and many children never learn to write properly. As if this weren't enough, a child of about eight or nine years has to learn to write all over again with cursive or 'joined-up' writing.

Compare this process with learning to touch-type. You must learn to recognise the different letter shapes and their positions on the keyboard, and connect certain keys with specific fingers. This is much easier, though some people might argue that the challenge of learning to write is valuable and should not be ignored. Yet mastering handwriting is often a barrier to written expression. If a group of children was allotted a certain amount of time to produce a piece of work,



Artistic Differences
These three drawings were all done by Timothy Ginn, aged 8. They all share several common features: fine attention to detail, shading to discriminate separate objects, and accurate placement and proportions of objects, for example. Yet, after a short time learning the mouse and Macpaint, Timothy created a drawing on the Apple Macintosh that makes several significant leaps artistically: perspective, as the ship is shown in three dimensions; a background, which is missing from the two freehand drawings; and movement, indicated by the particle trail that follows the ship, which is seen going away from us. Such differences as these can be even greater for children with learning disabilities: often they can visualise a scene, but are unable to draw what their mind can see

each child's output would depend on his handwriting abilities. We would expect only a line or two from a five-year-old, three or four lines from a six-year-old, and perhaps a page from a seven-year-old. If children were taught to type, teachers (and parents) might be surprised at the quality and quantity of their results.

Handwriting has not, of course, been made obsolete by the word processor, but in later life common practice increasingly dictates the use of a keyboard, both in business and social situations. Keyboard skills are often taught only to young people on business courses and the extension of these skills to children at school is beneficial. This trend will continue as computers become more common and their uses more diverse.

DESIGN WORK

Another aspect of microcomputing relevant to children is the increasing ease with which a computer can be used to create sophisticated graphics. This role of the micro has been seriously restricted because of cost — processing graphics generally requires more sophisticated and more expensive hardware than processing text. The economics of school computing are changing, however, as machines are simultaneously becoming more powerful and less expensive.

Initially, the graphics tablet (see pages 629 and 830) allowed various artistic functions to be used on the monitor, but recently these features have been incorporated into 'mouse technology' on, among other machines, the Apple Macintosh. By allowing the selection of 'icons', the mouse can give the user precise and immediate control over a vast range of activities, from drawing pictures and shapes to shrinking and expanding images, filling in areas with patterns and otherwise manipulating the display.

Furthermore, the computer allows the student to integrate and relate different material, textual and graphic, more easily than with traditional methods. A finished picture can be 'cut out' and 'pasted' into a passage of text previously generated. Although graphics programs such as Macpaint cannot replace artistic activities already carried out in the classroom, they offer a new medium with new techniques for children on which they can experiment, learn and express themselves.

Children want to learn. Their innate curiosity compels them to spend each waking hour searching, exploring and learning as much as they can about their new surroundings, and expressing their impressions through a creative impulse. Pencils and crayons are eagerly taken up, paper or the nearest available wall is drawn upon — and thus fundamental skills are developed.

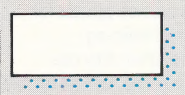
Two-year-olds with access to a computer make their investigations with the same exhilaration, pressing all the keys and watching little shapes appear on the screen. They accept new technology more easily than adults and, presented with an environment containing computers, the child uses them easily and naturally. It is only a matter of time now before schools incorporate micros as vital educational tools rather than regarding them as merely expensive toys.

VARIABLE TYPES

Symbolic Shapes
These three shapes are the symbols commonly used in PASCAL syntax diagrams:

- The rounded, or lozenge-shaped, symbol represents PASCAL reserved words, or characters that need no further explanation (such as letter' or 'digit')

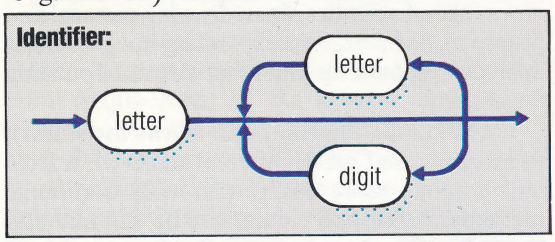
- A circle represents a PASCAL operator (+, —, *, ., etc)



- The rectangular symbol stands for a word or phrase that has its own separate syntax diagram

PASCAL uses four simple data types in its variable declaration statements. Here we discuss the differences between integer, real, character and boolean variables, giving examples of how these are used, and look at compound statements.

PASCAL provides four simple data types that are pre-defined for us, and given the identifiers Integer, Real, Char and Boolean. Numbers are classified according to whether they may have a fractional part (real numbers) or are natural whole numbers (integers). Of course, the actual range of numbers available is determined by how many bytes are used to store a given value of either type. Design decisions like this are taken by the compiler writer (the 'implementer') and such features are said to be 'implementation dependent'. All these implementation dependencies must be specified in the documentation for a compiler to be validated by the ISO (International Standards Organisation) standard for PASCAL.



The range of integers on your compiler will almost certainly be either from −32,768 to 32,767 or from −2,147,483,648 to 2,147,483,647, depending on whether a two or four byte representation is used. PASCAL names the value of the maximum integer, represented by the predefined constant identifier, MaxInt. You can therefore easily find this value with the statement:

```
WriteLn ('MaxInt is :',MaxInt)
```

Real numbers will also be held to a limited range and accuracy — usually from about 1.7E+38 — with six or seven digit precision at worst. This form of writing real numbers, known as 'scientific notation', is the default in PASCAL, but you may use the more usual form of writing them with a decimal point (e.g.123.456) if appropriate.

Be warned, however, that a real number *always* has a decimal point separating the whole and fractional parts, *both* of which *must* be present. So 0.1 and 1.0E−1 are acceptable, but .1 or 1E−1 are illegal. Fortunately, these strict rules only apply to numbers that must be recognised as real in the program text. If you are entering data from the keyboard, for example, an integer will be read and converted automatically if a real value is expected.

Char is short for character, of course, and a value of this type will be one of the members of the character set (usually ASCII) available on the computer. PASCAL ensures its own portability by assuming that:

- The characters A to Z will be alphabetically ordered, which means A is less in character value than B, and B less than C, etc.
- The digit characters 0 to 9 will be ordered and contiguous, meaning that whatever the value of 0, 1 will be the next, and so on.

ASCII character sets also have a contiguous alphabet, but this is not essential to PASCAL — merely convenient for PASCAL programmers! Each character value will have a numeric code, which is one value of a sub-range of the integer type. ASCII codes are defined over the range 0 to 127, and many machines will extend this to 225 for extra codes for graphics characters. We can easily map any ordered character set onto the scale of 'ordinal' values used internally by the computer. PASCAL provides the predefined function Ord: this returns the integer code of its argument — so Ord (A) is the equivalent of 65 in the ASCII character set. Another function, chr, gives the reverse mapping — chr (65) giving the character A (notice that a dollar sign is *not* used with chr).

The range of both character values and integers is defined for every implementation, and they exist on an ordered scale of known constants. For this reason they are termed 'ordinal' or 'scalar' types. Whatever the value, we always know what the previous and next values are, if any. These adjacent values can be obtained by the two scalar functions:

```
pred (item) (predecessor)
succ (item) (successor)
```

Thus succ (3) will always be the character value 4, but pred (Z) will only be Y on some character sets, such as ASCII. Pred (MaxInt) will be either 32,766 or 2,147,483,646. The chr function may only be used with an argument that is a character code. All the other scalar functions may be used with any scalar type, though if ord is used with integers, it just returns the value of its argument.

Boolean variables are the simplest of all the scalar types, as there are only two values in the scale — false and true (in that order). As they are simple scalar types, the scalar functions may be applied to any boolean value — the ordinal value of false is 0 and ord (true) is 1. The other scalar functions, pred and succ, however, aren't very useful here. Your PASCAL compiler should object

strongly if you try something like WriteLn (pred(false)) — it is hardly suprising to learn that it is an error to attempt to evaluate a nonexistent value.

The following constant definition part of a program shows all the simple ordinal types as they may appear in PASCAL source text.

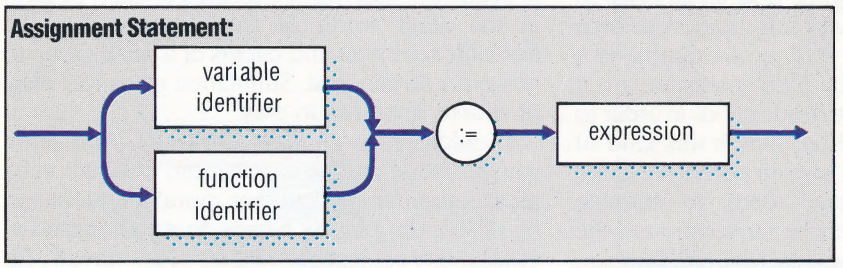
```
CONST
   VAT          = 0.15;
   columns      = 40;
   space        = ' ';
   debugging    = false;
```

EQUALITY AND ASSIGNMENT

The equals symbol (=) *always* means equals in PASCAL, and is used for equating constant identifiers to the values they will retain. When we declare variables in the VAR declaration section, the colon (:) separates the newly declared variable identifier from its type. For example:

```
VAR
   ratio        : real;
   number       : integer;
   symbol       : char;
   done         : boolean;
```

When we wish to assign values to these variables, the composite 'assignment operator' (:=) is used. This helps to distinguish clearly between the three



LIZ DIXON

classes of operation. CONST definitions equate permanent values, VAR declarations only reserve memory space, and assignment gives a (possibly temporary) value to the identifier.

THE COMPOUND STATEMENT

When two or more statements must be executed as part of a single process, we can bracket them between the words BEGIN and END as a 'compound' statement. Remember that each constituent statement must be separated from any following statement with a semi-colon. We have already seen a compound statement of course, since every program's body of executable statements takes this form. Here is a complete program that uses many of the features we have already considered. We will adopt the convention of writing the reserved words in upper case letters to help distinguish them from identifiers. Notice the blank lines left between each part of the program structure to aid readability:

```
PROGRAM Circle (input, output);

CONST
   pi           = 3.1415926536;
   prompt       ='Enter the radius:';

VAR
   radius,
   area         : real;

BEGIN
   WriteLn;
   write (prompt);
   read (radius);
   area := pi * radius * radius;
   WriteLn;
   WriteLn ('The area of a circle',
      'of radius', radius : 8 : 3);
   WriteLn ('is : ', area : 10 : 3)
END.
```

There are two aspects of syntax worth noting in this example. Firstly, the VAR part declares two identifiers of the same type — both real. Two separate declarations are not necessary, however, since lists of items are simply separated by a comma; this is universal in PASCAL. So, when we specify more than one argument to a procedure — as in the WriteLn statements — the same syntax applies. The second new feature is the output formatting used to override the default scientific notation of real values. We may optionally specify two integers, separated by colons, to force a certain field width for the entire number and its fractional part.

In our Circle program, three decimal places will be given in each case for both the radius and area. Because the area will be a larger (and therefore longer) number, we allow a total of 10 character positions instead of only eight for the radius. These integer values must exceed zero, allow for a possible sign, have at least one digit, and accommodate the decimal point to be written before the fractional part. Illegal values will cause errors at run time, or (at best) revert the format to scientific notation; WriteLn (X : 6 : 2) would not allow room for numbers greater than 99.99, for instance.

More valuable still, PASCAL will automatically round off the last digit for us to give the best accuracy within any requested numeric field. Further, any variable or expression may be used, not just a constant. This allows enormous flexibility, including tabulation facilities. With all other data types, only one integer value is needed — or indeed allowed — to specify the field width. To specify a width of one will cause integers to be written in the minimum size field, with no spaces. Therefore, we must remember to put them in ourselves if results are to be tabulated. For example:

```
WriteLn ('Total : ' : 20, weight : 1, 'tons.')
```

PASCAL will normally refuse to give a misleading or inaccurate output, but the ability to suppress all spaces means that we must be careful not to print two consecutive numbers with a field of one. For example, if 12 and 34 were written this way, 1234 would be the output.

TRADING PLACES

As well as being fun to play, strategy-based games that simulate real-life predicaments test the planning skills of an individual or a team and — uniquely — involve a moral dimension. Here we introduce a new BASIC programming project: constructing a trading simulation game based around a 15th-century voyage to America.

As well as providing fast-moving, reflex-testing games for 'shoot-em-up' fans, the computer is an excellent medium for a quite different type of challenge — the simulation game. This calls on skills very different from those required for arcade games. As the word 'simulation' implies, a computer-based model of the real world is created for the purposes of the game. These models may be 'real-time' simulators, such as flight simulation programs, or strategy-based.

In the strategy-based kind the player, or group of players, is usually assigned a series of tasks to perform and provided with information to help him arrive at decisions. A typical example is a trading game, in which the participants are given sums of money and have to trade goods in order to maximise their profits. What makes this kind of simulation different from, say, an adventure game is that, while the player tends to stumble across pitfalls and perils as he moves around the adventure world and deals with them as they arise, the player in a simulation game can often plan ahead, managing his resources to cope with expected (and unexpected) contingencies.

This type of game therefore demands of its participants management skills and foresight; in other words, the ability to plan ahead rather than 'on their feet'. To reflect the real world, of course, not everything in a simulation game should be allowed to go exactly according to plan — a good simulation will always introduce random factors to upset the players' schemes. A good player must therefore involve himself in contingency planning to minimise the damage inflicted by unpredictable events.

Simulation games have the advantage of allowing people to play together as a team, combining their wit and judgement against the computer to reach a predetermined goal. For this reason, simulation games are beginning to be used in schools, as they allow children to model the sort of behaviour that will be expected of many of them in the adult world of work, husbanding the available resources and talents of a small group to achieve a certain goal. Simulation games are also, of course, great fun to play.

In addition to all these attributes, simulation games extend into an area seldom touched on by most computer applications: moral questions can be raised, the players having to weigh increased profitability against the welfare of employees, for example. When such decisions have to be made in committee, they are often preceded by heated debates on the rights and wrongs.

Many different scenarios lend themselves well to simulation by computer, the common factor often being that the simulation involves performing a specific task to achieve a specific goal. In some cases these can be selected by the player. It may be that the goal is to stay in business for a certain length of time, or alternatively, to make a million pounds. Many simulations are based around financial strategy (see page 401) but some have more philanthropic goals. One simulation game currently used in primary schools is a project to locate and raise the wreck of the Elizabethan galleon, the *Mary Rose*. In this game, players are also given documents to supplement the program and provide additional clues to assist the decision-making process.

In this programming project series, we'll be constructing a trading simulation game, putting the players in the position of having to equip and sail a ship to the New World in the 15th century. The game is structured as a series of modules that fit together, and the program is designed so that

Worlds Of Make-Believe
Shown here are examples of the three main types of simulation. Space Shuttle is a realistic arcade-type simulation of the real spacecraft. Air Traffic Control is a real-time simulation, in which you must give instructions to various aircraft to enable them to take off and land safely. The Great Nordic War is a war game simulation of the Thirty Years War, with the player taking on the part of Charles XII of Sweden

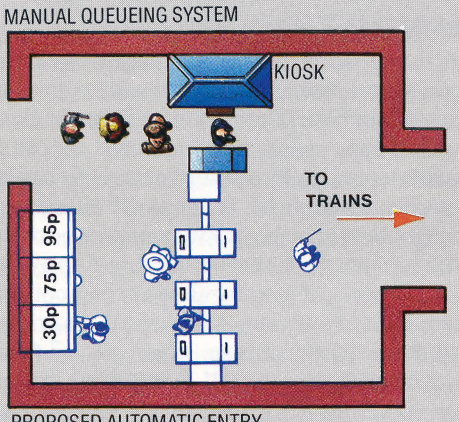
LIZ HEANEY

Going Underground

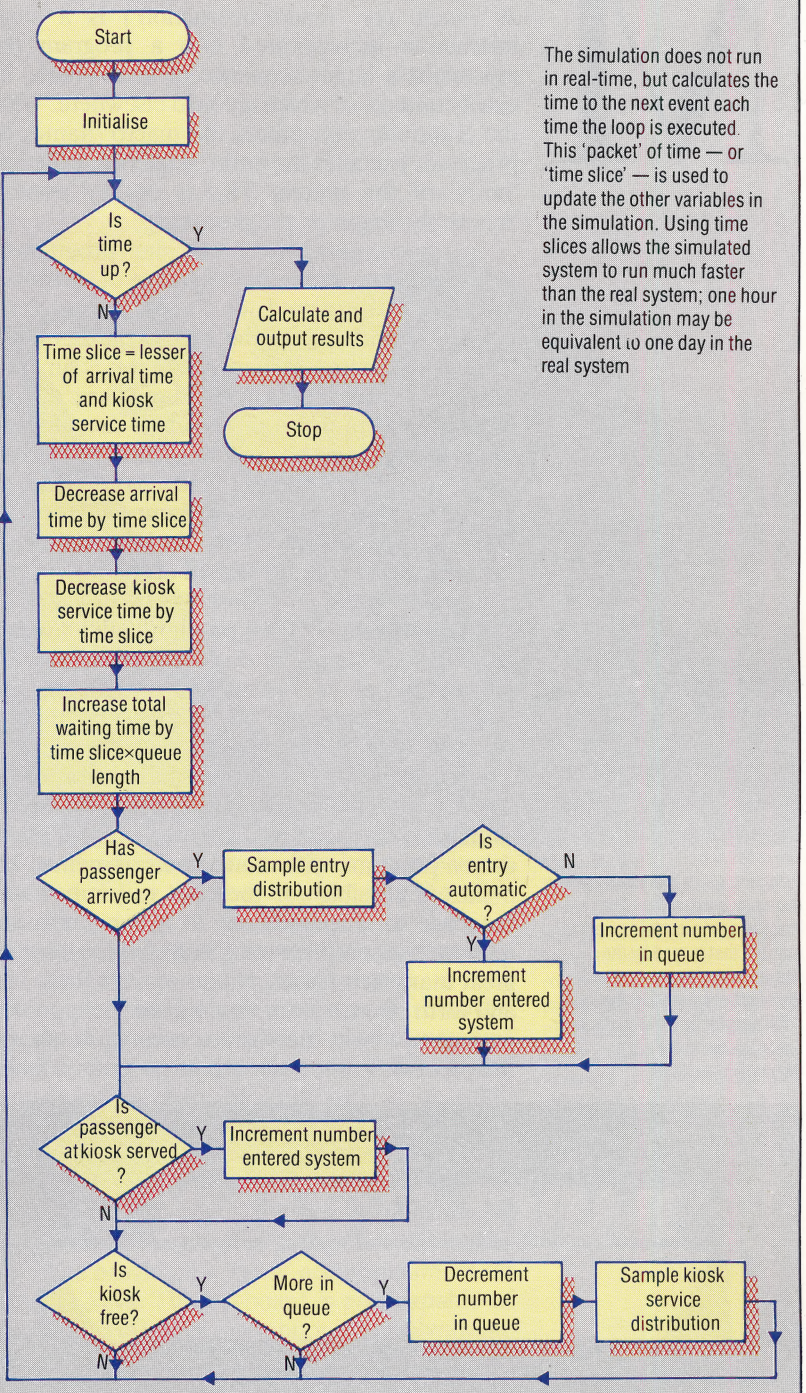
Simulations are now widely used in education and for games, but computer-based simulation was first developed as a design and management tool. Simulations are of great use in the design of traffic flow and service systems, where the nature of the model makes it ideal for computer simulation. Such simulations are usually based around statistical and mathematical principles that predict frequency distributions of the variables used.

We can demonstrate the principles of a simulation by looking at a simple service system. The existing system allows entry to the underground station by buying a ticket at a kiosk. It is proposed to install a number of automatic ticket barriers that are operated by pre-paid tickets. The flowchart illustrates the processes involved in producing a simple simulation of the system. Three distributions are used: a passenger arrival distribution, a kiosk service time distribution, and a method of entry (i.e. manual or automatic) distribution.

The first two distributions are derived from statistical observations and mathematical methods; the third will be varied on each run of the program in order to see the ramifications of various proportions of manual and automatic entry gates. This simple simulation will calculate the average waiting time of passengers entering the system



PROPOSED AUTOMATIC ENTRY SYSTEM



The simulation does not run in real-time, but calculates the time to the next event each time the loop is executed. This 'packet' of time — or 'time slice' — is used to update the other variables in the simulation. Using time slices allows the simulated system to run much faster than the real system; one hour in the simulation may be equivalent to one day in the real system

the scenario can be easily changed, providing a skeleton around which a different context could be built. Thus, the simulation could be updated to become a science-fiction journey, trading with a distant planet by spaceship.

The object of the game, for one or more players, is to trade goods to maximise profit, but balancing this aim against the safety and well-being of the crew. The game will be divided into three stages: preparation, the journey and trading. The first phase involves gathering resources and planning the voyage, and the second introduces random elements into the game. Your ability to deal with these contingencies will largely depend on how well you planned during the first phase of the game. The third phase involves establishing good relations with the local inhabitants and trading profitably with them.

From a programmer's point of view the construction of the game will be as a series of independent modules, and BASIC flavours will be given for the BBC Micro, Spectrum and Commodore 64. The main tasks of the program are to keep track of the players' decisions, create various contingencies and analyse and monitor the player's status as the game proceeds. In the next instalment we shall tackle the first program module for the game.

M

MACRO

A *macro* (or 'macro-instruction') is a single instruction incorporated into a program that represents a larger sequence of instructions, the exact contents of which are held elsewhere. When the computer encounters the macro 'name', the processor executes the entire sequence, called the 'body' of the macro. This is usually done in Assembly language programming.

A 'macro-assembler' lets the programmer define a macro, listing the sequence of operations as they are to be carried out. The macro then forms a single command in the text of an Assembly language program. In many ways, this is similar to defining a procedure in a structured BASIC or LOGO program except that a procedure is executed as a subroutine; when the assembler meets the macro name, however, it copies the macro code into the program, replacing the macro name with the code. Once a program procedure has been defined, it can be called by name and all of the instructions that make up the procedure will be executed.

Some computers have programmable function keys, which operate in much the same way as a macro. A set of operations is defined, then assigned to a particular key so that every time the key is pressed, the body of instructions is carried out. Some software developers have incorporated this concept into applications programs, such as Lotus Development Corporation's 1-2-3 menu-driven program, with many levels of menus. This makes it possible for the user to jump from a database to a spreadsheet to a bar graph very quickly, but complicated operations can involve a great many keystrokes. If you want to jump back and forth often, a lot of time will be spent pressing commands from the various menus. To shorten the process, Lotus built into its program a facility known as the 'keyboard' macro. A single key can be labelled with the name given to a lengthy set of instructions. When that key is pressed, in combination with a special function key, 1-2-3 looks to the location of the macro body and executes the keystrokes defined. Lotus refers to this process as 'automating' keystrokes.

Plastic Money
The most popular use of magnetic cards is in bank cash dispensing machines. The magnetic strip contains a pass number that is read by the computer and compared with the number entered by the user. If the number is correct, the user gains access to the cash dispensing facilities

MAGNETIC CARD

A *magnetic card* is a data storage medium of heavy paper or plastic card, like a punch card, which is partly coated with a magnetic recording material. This material is similar to the magnetic surface on cassettes and disks. Magnetic cards were a prominent storage medium for data processing and word processing in the late 1960s, used particularly by IBM in tandem with its Selectric typewriters. The magnetic cards were used to store data, after which time a card reader sent the stored information back through the typewriter to be printed.

The primary use for magnetic card memory today is the bank or credit card. A magnetic strip along the back of the card records details such as the card owner's name, account number and bank reference. Bank machines use the recorded information to process your machine transactions. Many retailers have card readers next to their cash registers, so that a credit card can be automatically checked against lists of lost and stolen cards, and instant approval for the purchase can be given, via a modem and phone, by the central credit computer.

MAGNETIC DISK

A *magnetic disk* is a data storage device consisting of a round plate, its surface coated with a magnetic recording film. The plate itself can be flexible, as with floppy disks, in which case the underlying material is usually a soft plastic. Or it can be rigid, usually made from an alloy of aluminium. The magnetic film can also vary in composition, from the ferromagnetic oxide used on floppy disks to a thin coating of a metal alloy such as cobalt/nickel or cobalt/chromium. Magnetic disks on computer systems vary in size from the 7.6cm (3in) microfloppies to the 91.4cm (36in) devices used on some large mainframes.

MAIL BOX

A *mail box* is storage space in computer memory for information that has been transmitted from one computer to another on an 'electronic mail' system. The transmitting station affixes a unique address to the letter or other document — the address being the mailbox, where the data is held until the addressee recovers it. A mailbox is usually secure, and the electronic mail system requires the addressee to enter a password before the letter can be retrieved. If a letter is found after the user has logged on to the system and checked his mailbox, it can be downloaded onto the user's system, where it can then be printed or saved to disk.

TUNING UP

Yamaha has long been a producer of high quality musical instruments, including electronic synthesisers. The recently released MSX-standard CX5M micro takes advantage of the company's expertise in this field, and is the first home computer dedicated to music-making. We sound the machine out.

Although adhering to the MSX standard, which enables it to be used for applications such as games playing or word processing, the Yamaha CX5M computer is aimed squarely at the electronic music enthusiast. A YK-10 or YK-01 external piano keyboard can be plugged into a socket on the side of the machine, where there is also a pair of MIDI (Musical Instrument Digital Interface) ports, enabling any synthesiser or electronic music device with a suitable MIDI interface to be run from the computer. There is also a pair of microjack sockets that will accomodate the use of external speakers.

The computer looks very similar to other machines in the MSX range. There are 48 typewriter keys surrounded by 10 control keys, containing such MSX-standard features as a destructive backspace, a Graph key to display a set of graphics characters, and a Code key, which, when pressed with one of the typewriter keys, prints a variety of foreign language characters. Above the keyboard are five function keys, which are capable of providing 10 programmable functions. On power-up these default to the MSX-standard functions of AUTO, LIST, RUN and so on. The bottom right-hand corner of the keyboard frames the four-key cursor cluster, and above this is a set of five keys featuring other MSX standard commands such as INSert, DELete and STOP. At the top of the keyboard is a cartridge port.

MACHINE ARCHITECTURE

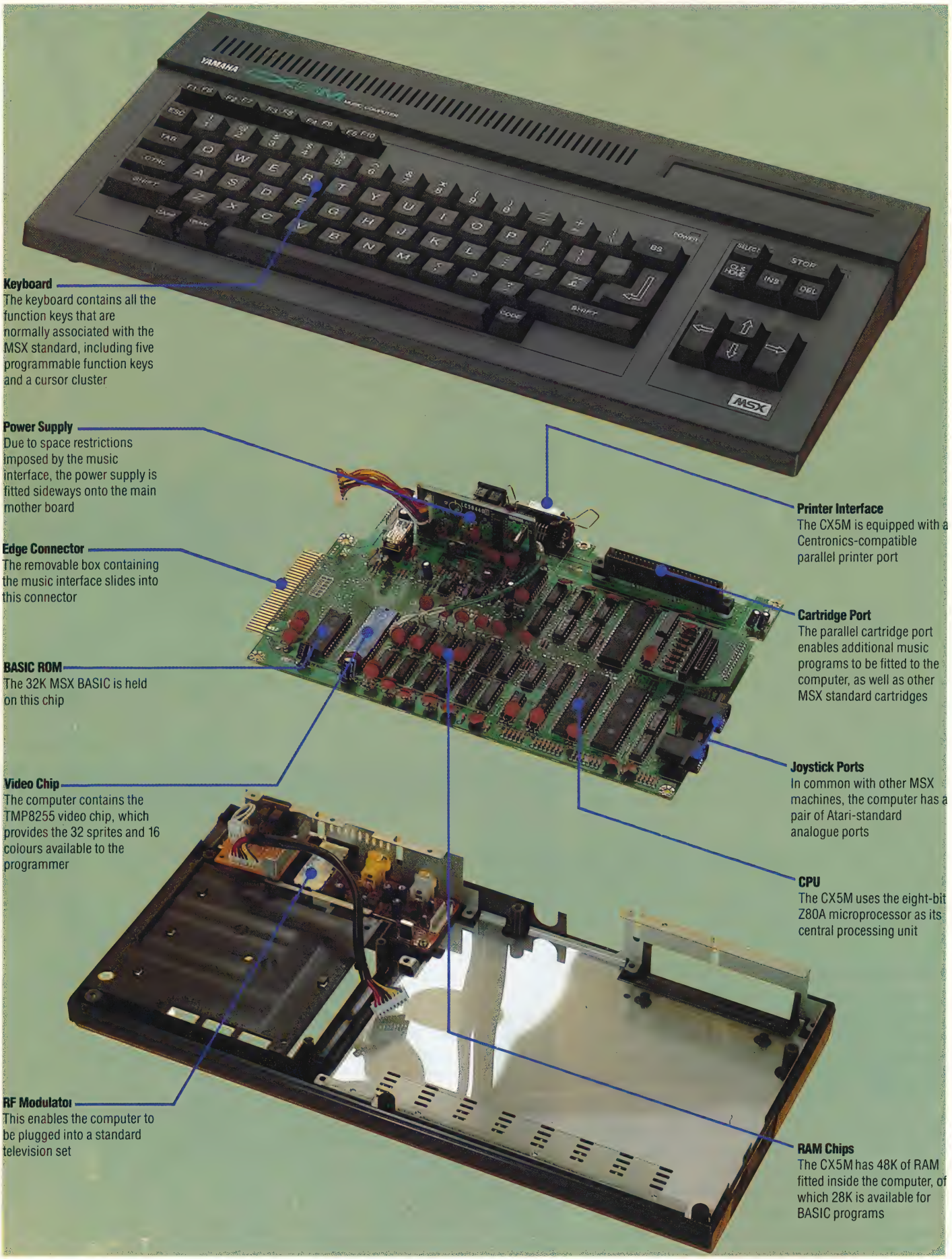
On the right-hand side of the CX5M is a pair of Atari standard joystick ports. The back of the computer contains a parallel edge connector, the Centronics compatible printer interface, a cassette port, a sound jack for external mono output, a composite video monitor jack, an RF television jack and the power supply input. The MIDI ports and keyboard interface, which give the CX5M its unique musical capabilities, are contained within a single box. This slides into an edge connector underneath the machine on the left-hand side. The reason for this is that Yamaha intends to produce a series of interfaces with different configurations. When the devices become available, users will be able to swap interfaces merely by removing a screw and sliding the new interface into the edge connector.

CHRIS STEVENS

Music Boxes
The Yamaha CX5M is expected to sell better through musical instrument outlets rather than computer retailers. To emphasise its musical capabilities the computer is marketed as a package with either a YK-01 or YK-10 piano keyboard. The piano keyboard plugs into a small music interface box that is screwed on the underside of the computer

The three and a half octave YK-01 keyboard — sold with the computer, and shown in our photograph — has a nicely professional feel to the keys, although serious keyboard players may find them too small to be entirely comfortable. Using the on-board software, the keyboard can be 'split', which means that one programmed voice can be played on the lower part of the keyboard while another completely different voice can be played with the rest of the keys. This means, for example, that you could be playing 'strings' on the higher keys while providing a 'brass' accompaniment on the lower keys. The keyboard is also capable of being played in either monophonic or polyphonic mode (or both), with up to eight notes being played simultaneously.

On power-up the screen displays the MSX-standard blue screen. The music program can be accessed by the CALL MUSIC command, which will display a menu containing five blocks of the various options available. The user can then move down the list by using the Return key, and the parameters of each option are altered by pressing the cursor keys. The blocks labelled POLY and MONO enable the player to select which of the available 46 pre-programmed voices are to be used, and in which of the two modes. These voices range from conventional musical instruments — such as the organ, guitar and percussive instruments like the vibraphone and cowbell — to a range of everyday sounds — such as an



Keyboard
The keyboard contains all the function keys that are normally associated with the MSX standard, including five programmable function keys and a cursor cluster

Power Supply
Due to space restrictions imposed by the music interface, the power supply is fitted sideways onto the main mother board

Edge Connector
The removable box containing the music interface slides into this connector

BASIC ROM
The 32K MSX BASIC is held on this chip

Video Chip
The computer contains the TMP8255 video chip, which provides the 32 sprites and 16 colours available to the programmer

RF Modulator
This enables the computer to be plugged into a standard television set

Printer Interface
The CX5M is equipped with a Centronics-compatible parallel printer port

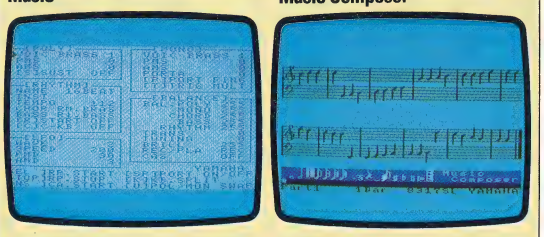
Cartridge Port
The parallel cartridge port enables additional music programs to be fitted to the computer, as well as other MSX standard cartridges

Joystick Ports
In common with other MSX machines, the computer has a pair of Atari-standard analogue ports

CPU
The CX5M uses the eight-bit Z80A microprocessor as its central processing unit

RAM Chips
The CX5M has 48K of RAM fitted inside the computer, of which 28K is available for BASIC programs

These are some of the cartridges that are currently available for the CX5M. Each of the programs shown here allows the computer to perform a different function — from programming the DX-7 synthesiser to composing music programs. Each cartridge comes with its own manual. However, at around £35 each, they are relatively expensive and users may prefer to wait until the price comes down or the software improves



IAN McKINNELL

Shown here are two of the applications that are available for the Yamaha CX5M. The Music package is held on board the computer and is accessed via a CALL MUSIC command. By moving the cursor around with the Function and Return keys, the parameters shown can then be altered by pressing the cursor keys. The Music Composer allows the player to write a musical score on the screen that can be edited and then sent out to the printer to produce 'hard copy'

'ambulance siren' and 'sound of raindrops'. Due to the nature of some voices, they are unavailable when 'sustain' is being used. Altering the voices of the POLY and MONO modes enables the user to set contrasting tones for use with a split keyboard — that is, designating some keys as monophonic, others as polyphonic. It is possible, of course, to set the entire keyboard in either mode; it is not, however, possible to split the keyboard into two POLY sections.

The sounds that can be produced are remarkable, even when transmitted through an ordinary television speaker, although some of the sounds bear little resemblance to their given names. For example, 'train' sounds merely like an ordinary electric organ. This is a common fault of synthesiser manufacturers, but providing the user doesn't expect an exact reproduction of an everyday sound, most players will be more than happy with the voices provided.

Another block, RHYTHM, provides a rhythmic accompaniment to a melody. There are six different rhythms available that can be constructed using chords, bass, and percussion. Of these, the bass and the chord tones can themselves be set by the player. The tempo of the rhythm can be varied and there is also an option allowing the player to vary the pitch of the rhythm from the piano keyboard. By altering the shape of the sound wave with the LFO (Low Frequency Oscillator) block, tones can be modulated to produce different sounds. The BALANCE block allows independent volume setting of each rhythmic element, plus the MONO and POLY modes. Thus, the rhythmic accompaniment can be lowered into the background and the volume of the POLY mode raised to give added emphasis to the melody. Once the player is satisfied with the sounds that are produced, the data can be SAVEd onto cassette and LOADed again later. This saves the player having to write down all of the various parameters that have been selected and then resetting them.

Apart from the on-board software, Yamaha has launched a series of cartridges to be used with the CX5M. This range of cartridges includes the FM Voicing Program, which is an extended version of the on-board software. The program allows the player to more fully manipulate the shape of a voice by adjusting the frequencies and algorithms with which it was constructed. A second voicing program on cartridge allows the computer to program the highly rated Yamaha DX7 synthesiser. The FM Music Macro enables the computer to be programmed with a set of additional BASIC commands, whereas the FM Music Composer displays a blank stave and allows a conventional score to be entered and then dumped to an external printer.

Although it might seem that Yamaha has provided a comprehensive suite of programs for the CX5M, one is left with the impression that the software does not fully exploit the capabilities of the machine. The modulation programs, in particular, seem a little limited — at times it is hard to discern any noticeable change in the sound of a note. This is unusual, as Yamaha has based the CX5M operating system on that used with the DX7 synthesiser — its enormous range of sounds makes the CX5M look puny by comparison. Futhermore, although the methods chosen by Yamaha to alter the parameters of the blocks are adequate, they do at times seem cumbersome — for example, using cursor keys to alter parameters while moving the cursor with the Return key. However, Yamaha says that updated software is on the way.

The CX5M is obviously aimed at people who want a home computer and who also have an interest in electronic music. For those people the CX5M at around £500 is obviously a very tempting package. Not only do you get a complete computer MIDI system and a keyboard, the CX5M offers much more potential than a similarly priced system based around, say, the Commodore 64. It would appear, however, that the CX5M is destined to become a cult computer. It is difficult to justify the outlay to someone who is not interested in electronic music, since they can now buy a similar MSX computer for less than half the price. How successful Yamaha is in establishing the computer as the machine to buy for music enthusiasts depends on the future development of its software and music interfaces.

YAMAHA CX5M

PRICE
With YK-01 keyboard: £534; with YK-10 keyboard: £614 (both inc VAT)

DIMENSIONS
413 x 216 x 64mm

CPU
Z80A, running at 3.58 MHz

MEMORY
48K RAM of which 28K is available for BASIC programs

SCREEN
40 x 24 text screen; 256 x 192 pixel graphics screen, with 16 colours and up to 32 sprites

INTERFACES
Centronics printer, TV, composite monitor, audio output, stereo output, 2 joystick ports, cassette port, ROM cartridge port, expansion bus, keyboard interface, MIDI input and output ports

LANGUAGES AVAILABLE
BASIC, PASCAL, Assembly

KEYBOARD
67 key typewriter-style with cursor cluster, plus 5 programmable function keys. The piano keyboard is eight note polyphonic with a range of 3.5 octaves

DOCUMENTATION
The manual is very patchy. While giving a few details on how to use the music software, there is no tutorial explanation given. MSX BASIC is hardly mentioned

STRENGTHS
The machine is excellent value for the electronic musician. Viewed as a musical instrument, the CX5M has no other competition in the price range

WEAKNESS
The software will need to be greatly improved if the machine is to enjoy long-term success. The manual is very limited and the machine is too expensive for anyone not interested in electronic music

BIBLE READING

"The Word" Processor
For MS-DOS machines
Price:
£254.25 (including VAT and P&P)
Distributors:
Access Software Ltd,
36 Aybrook Street, London W1M 3JL
Authors:
Bible Research Systems, Austin, Texas
Format:
Disks

"The Word" Processor is a program designed for the Bible student, employing full search and indexing capabilities. While obviously a highly specialised package, it reveals some interesting programming techniques that could have widespread application elsewhere.

Certain occupations and scholastic studies require constant searching through large bodies of text. Bible students, for example, require concordances (specialised indices) to help them study the 39 books of the Old Testament and the 27 books of the New Testament — approximately 750,000 words in all.

Computers, generally, come into their own when such large quantities of text need to be searched, but the limitations of memory and disk capacities often constrain this application in home micros. Obviously then, a specifically written program could make searching for appropriate Biblical texts a much simpler task. The light-heartedly named "The Word" Processor not only allows you to search for specific references, but also to construct indices, which cut down the amount of time a search can take.

The sheer magnitude of the task undertaken by the programmers of "The Word" Processor becomes apparent when the package is opened. The manual is a slim booklet of 36 pages, but the complete program, including Biblical text files, consists of seven double-sided disks, all of which need to be copied. If a 40-track, single-side drive is employed, you would then have to handle a total of 14 separate disks, immediately demonstrating the manipulation of so many 5¼in disks to be too cumbersome to be really efficient.

PROCESSING THE WORD

"The Word" Processor requires careful installation. It is reasonable to assume that most students are not necessarily computer buffs, yet the BASIC program is rather tricky in the way it is run: when we used it, it stopped responding to MS-DOS and would only respond to PC-DOS. The BASIC interpreter and the machine system tracks from the DOS disk must be copied onto the program disk. The student must then choose between RUNning the main program (TWP), if he is using an IBM PC, another version for IBM-compatibles (ATWP), or a special version for computers with only 64 Kbytes of memory (TWP64). The experienced user might reasonably assume that once the system tracks, BASIC and the relevant programs have been installed onto disk, it would then become autobooting, but there is no AUTOEXEC.BAT program for this purpose.

Once LOADed and RUNning the title page makes one realise that this is no ordinary software package, for in addition to the usual software protection notice ("This software is protected by USA copyright laws. Reproduction or use of unauthorised copies may result in imprisonment or fines up to $10,000"), there is also an appropriate Biblical quote:

EXODUS 20:15 "Thou shalt not steal."

This injunction is followed by an opening menu:

<F1> — DISPLAY HELP INSTRUCTIONS
<F2> — SET CONTROL OPTIONS
<F3> — PRINT MODE IS OFF. TURN IT ON
<F4> — RANGE:
<F5> — END THIS SESSION
<F6> — CREATE AN INDEX
<F7> — DISPLAY OR MODIFY INDEX
<F8> — MERGE INDEXES
<F9> — DELETE AN INDEX
<F10> — DISPLAY SCRIPTURE TEXT

<F2> produces a sub-menu of nine control options, including width of screen (40 or 80 characters), tabulations for the printer right and left margins, line-spacing, number of lines to a page, the maximum index size, display of one text at a time or an entire screen of text with verses shown in context, and the names of all indices available on the current disk.

The <F4> option is a way of managing both the vastness of the Bible and the proliferation of disks on a floppy-based system. Using a three-letter designation for each Biblical book — from 'Gen' to 'Rev' — and the chapter and verse reference, you can define the beginning and end of the search parameter. Should the user wish to read a specific book, the chapter and verse reference can be omitted and the program will assume that the beginning of the first chapter and the end of the last is the required range.

After the appropriate disk has been inserted, it is possible to browse through a chapter — displaying the first verse of the range with <F10> — and moving forward or back with the function keys, which are labelled across the bottom of the screen. Control option <F8> changes the display from one verse at a time to an entire screen of text, allowing a specific verse to be read in its context.

Search criteria can be set up with <F2>, the SCAN option. As with searches on most databases, it is possible to search for all occurrences, even within other words — so that 'man' would turn up

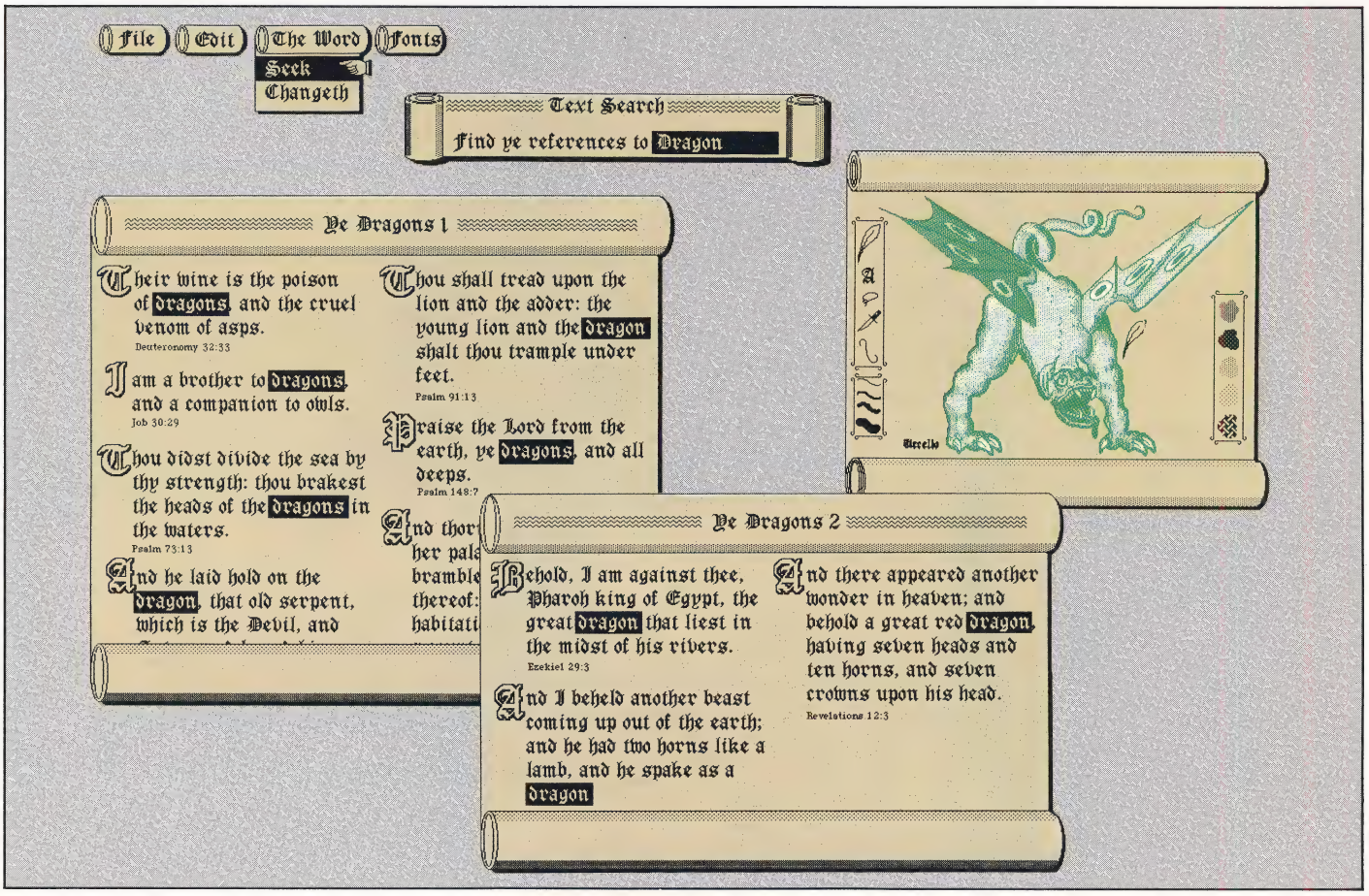
'demand', as well — or you can restrict the search to only the specific word named. 'Wild cards' are also possible, allowing all words beginning or ending with a specified search string to be found. Once set up, references to all criteria will be displayed one at a time, either in context or singly, depending upon what has been commanded.

Indices can be created with the <F6> option — either automatically or manually. Indices can also be edited (for example, deleting unwanted references), and two or more can be merged together — providing the limit of 1,020 on the number of items in an index is not exceeded. This limit can be increased to 53,040 with the <F2> SET CONTROL OPTIONS choice from the main menu. If an index has more then 1,020 references, it is stored (using the CONTROL OPTIONS sub-menu) in sections, and there will be a pause between the 1,020th and 1,021st entries while the next section of the index is READ from disk into memory. It is also possible to create an index of topics contained within each verse. This obviously has to be executed under the user's control, who defines the topic title and the verse within which it can be found.

Apart from the difficulty of manipulating such large amounts of text, the chief problem with "The Word" Processor is that since it is based on the 1611 King James Version, many of the words used in that superb translation have changed their meaning radically in the subsequent three and a half centuries. According to the translators of the 1952 Revised Standard Version, there are over 300 such words and phrases; and that's not including the errors in translation highlighted by the discovery of more accurate texts since the 17th century.

Fortunately, an associated program, The Greek Transliterator, can be used to avoid this pitfall, at least insofar as the New Testament is concerned. This allows you to search the Greek text looking for occurrences of a given English word in the King James Version, asking for the original Greek word to be displayed (in transliterated Roman characters, not Greek orthography) with a definition of the Greek word and, if necessary, how often each word was used and how it assumed different meanings in various contexts. Indices created with TWP can also be used with The Greek Transliterator, and indices of Greek words can also be created. The price of The Greek Transliterator is the same as for TWP.

To date, no transliterator has been produced for the Hebrew Old Testament, though a mainframe in Jerusalem has been programmed to hold the entire Talmud and other rabbinical works. This is possibly the most practical way of employing computerised Bible study: as a public access database, rather than as an individual text retrieval system. However, the techniques employed in "The Word" Processor are intriguing and could, presumably, be applied to other large-scale works: the plays of Shakespeare, for instance, or even the collected works of Marx and Engels.



King James And The Dragon
When queried on the keyword 'dragon', "The Word" Processor database located these and several other references. Acting as a written concordance would, "The Word" Processor simplifies searching for related Biblical ideas, symbols or personalities in the King James Version of the Bible

IAN McKINNELL

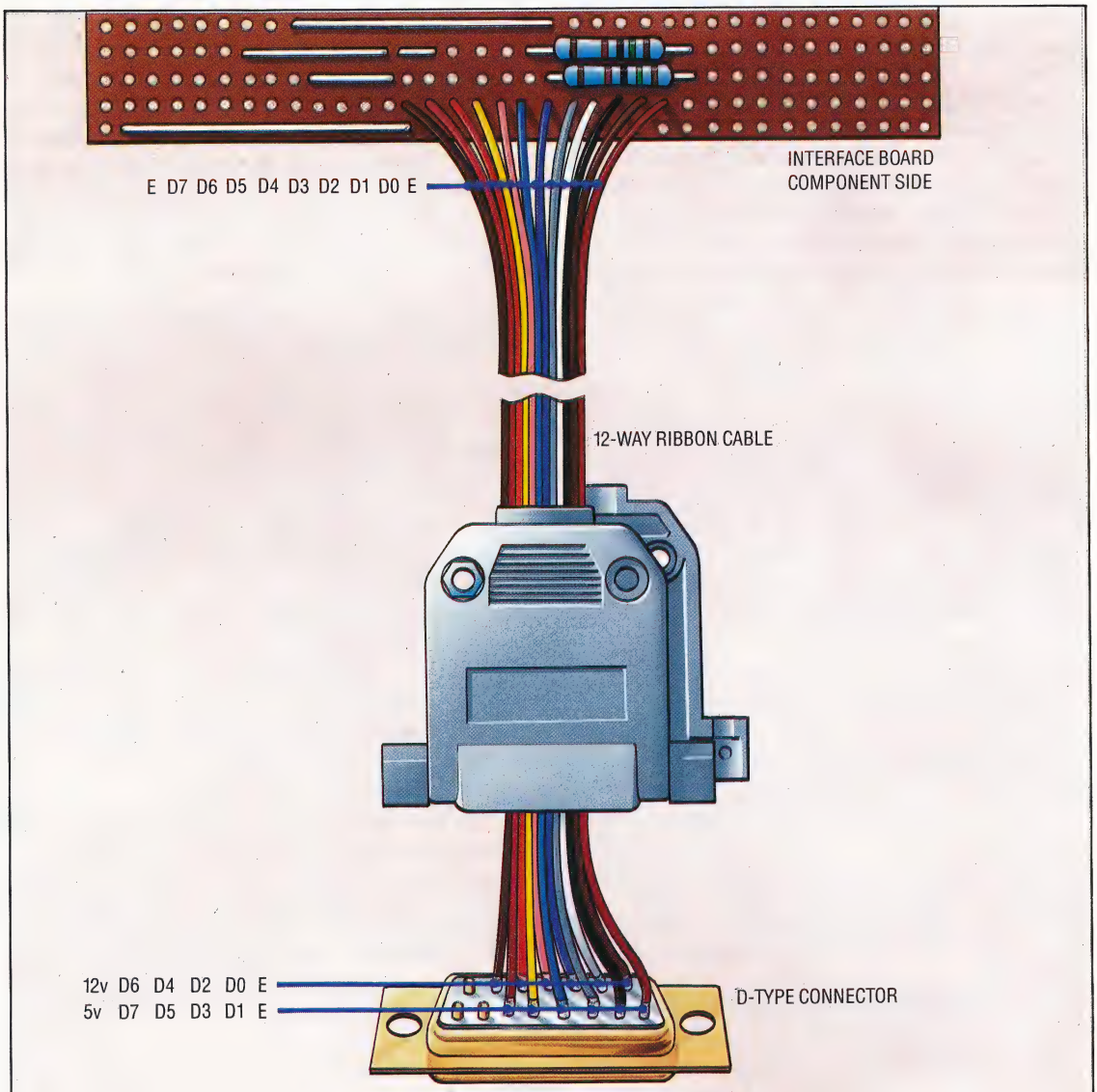
ON A LEAD

In the previous instalment we built an interface to allow the Workshop robot to be controlled from the Sinclair Spectrum. Here, we make up the lead that connects the robot to the interface and give Spectrum versions of some of the software we have already designed to control it.

The final construction task in building the interface is to solder a 12-way ribbon cable from the interface board to a D-type connector that will plug into the robot. The eight data lines from the mapped I/O port are available from the data bus via the interface circuitry. The 12v and 5v power feeds are also accessed directly from the Spectrum's expansion port. This means that we do not have to provide a further power source to drive the robot. The ZX power pack supplied with the Spectrum is capable of powering the computer and robot together.

Make the ribbon cable by stripping four wires off the ribbon cable specified in the parts list (see page 1012). Strip and tin both ends of each of the 12 strands that remain and solder each strand to the board and the D-type connector as shown in

Tie A 12-Way Ribbon
Wire up the D-type connector to the interface board as shown. The wiring is straightforward — wires alternating between the top and bottom rows of pins on the D-type connector



Interface/Robot Cable Connections

the diagram. Check your work thoroughly, ensuring particularly that the 12v and 5v power feeds run to the correct pins in the D-type connector.

The interface is now complete and we can plug it into the Spectrum to test it. With power off, carefully plug the interface board into the expansion port on the back of the Spectrum. The board should be inserted so that the component side is uppermost — if you have inserted the blanking plug into Position 5 of the expansion port connector it will be impossible to fit the board any other way. The board may be a tight fit and using a slight rocking motion, left and right, will help the expansion port PCB slot snuggly into the connector on the interface board. Plug the ribbon cable into the robot and switch on the power. If all is well the familiar Sinclair copyright message will appear on the screen. We are now in a position to try a test program.

MOTOR CONTROL

The robot maps into I/O port 31, its eight bits being used to control the motors and receive inputs from the robot sensors. The lower four bits control movement. Bits 1 and 2 control the direction of rotation of the two stepper motors used with the robot. All four directions can be selected by setting these two bits in various combinations. Bit 3 is the pulse bit. Changing this from 0 to 1 causes both motors to turn one step in which ever direction has been specified by the corresponding direction bit. Bit 0 is the reset bit and is normally set to 1.

We can test the output side of the interface by typing in and running the Robot Controller program, which allows the robot to be controlled from the keyboard. The letters T, B, F and H correspond to forwards, backwards, left and right, respectively.

The initialisation routine sets up four variables, corresponding to the four possible directions of the robot. Their values are set by the four combinations of bits 1 and 2 in I/O port 31. If we look at these variables and their four-bit binary values (see the table) we can see why.

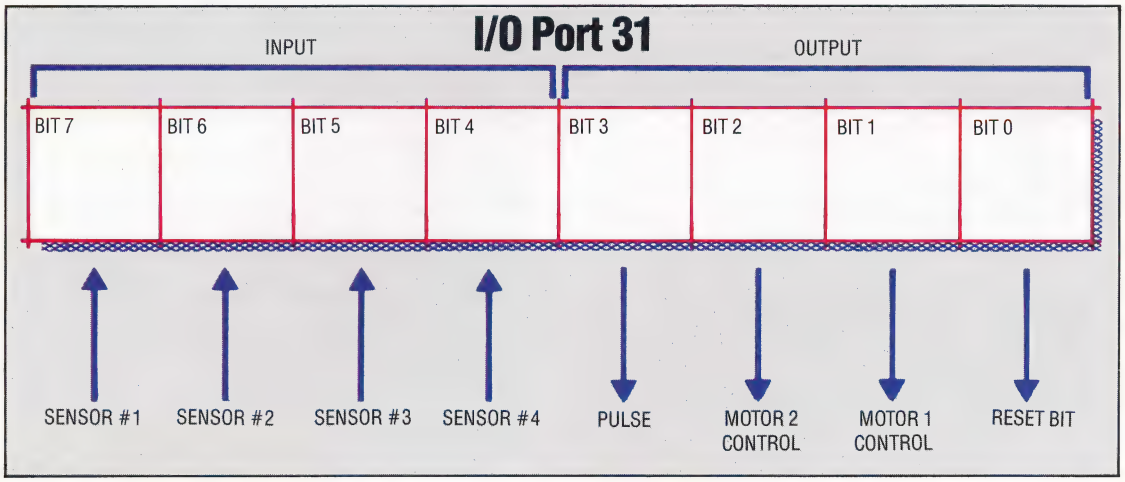
The direction taken by the robot is held in the variable dr. To make the robot move in the specified direction we must pulse the motors by setting bit 3 high and then low. The subroutine at line 2000 does this, using OUT, in a similar way to POKE, to place a value in I/O port 31. The value itself is first calculated as the current direction, dr, plus 8 to set bit 3 to 1, plus 1 to set bit 0 to 1, — that is dr+9. To turn bit 3 off we simply do not add in the 8, giving us dr+1. Causing bit 3 to go high and then low again makes the motors turn through one step of 7.5°. As this corresponds to a movement of less than 1mm at the wheel, the pulsing process is placed inside a loop to repeat it a set number of times, as determined by the variable m.

Direction	Variable	Decimal Value	Binary Value
Forwards	fw	4	0100
Backwards	bw	2	0010
Left	lf	6	0110
Right	rt	0	0000

Direction is altered by making a press on the keyboard, altering the value of dr. When the motors are next pulsed the motor direction bits are therefore changed, causing the robot to move in another direction.

DETECTING INPUT

The upper four bits are used to detect inputs from the robot's sensors, each bit corresponding to a socket in the patching system on the robot's lid. Normally these bits are held high (that is, they have a value of 1), unless they are grounded out by closing a switch, in which case the bit value changes to 0. To detect independent inputs to each bit we need a method of isolating each of the upper four bits and testing their values. In most dialects of BASIC this is possible using the logical AND instruction to mask off any bits we are not interested in. Spectrum BASIC has an AND instruction, but it is not sophisticated enough to mask individual bits in a number. The Z80 instruction set, however, contains an AND operation that will do the required task for us. We therefore need to write a simple piece of machine code that will perform the logical AND on two



Bits of The Action
The Spectrum interface sets the lower four bits of I/O port 31 for output and the upper four bits for input. The output bits control the stepper motor functions, the upper four bits being used to accept sensory input

numbers and return the result. To use the routine we must POKE the numbers we wish to AND together into two memory locations. The result is returned by the USR command. This roundabout method allows us to read each input line independently.

The Spectrum Bumpers program tests the action of the input side of the interface by sending the robot forwards until it meets an object and then retreats to its initial starting position (see page 895).

To bring Spectrum owners up to date we also include Spectrum flavours for some of the other programs devised in the robot application series. Readers should refer back to the page references given and make the necessary alteration to the Commodore 64 versions. Spectrum owners may also wish to substitute lower-case variables names for the upper-case names used by the Commodore versions.

Robot Controller Program

```
  10 REM **** spectrum robot controller ****
  20 GO SUB 1000: REM  initialise
  30 LET a$=INKEY$: IF a$<>"x" THEN
  GO SUB 3000: REM read key
  40 LET m=10: GO SUB 2000: REM pulse
  50 IF a$<>"x" THEN  GO TO 30
  60 OUT 31,0
  70 STOP
  80:
1000 REM **** initialise ****
1010 LET fw=4: LET bw=2: LET lf=6: LET rt=0
1020 LET dr=fw
1030 RETURN
1040:
2000 REM *** pulse ****
2010 FOR c=1  TO m
2020 OUT 31,dr+9
2030 OUT 31,dr+1
2040 NEXT c
2050 RETURN
2060:
3000 REM **** read keys ****
3010 IF a$="t" THEN  LET dr=fw: RETURN
3020 IF a$="b" THEN  LET dr=bw: RETURN
3030 IF a$="f" THEN  LET dr=lf: RETURN
3040 IF a$="h" THEN  LET dr=rt: RETURN
3050 RETURN
```

Spectrum Bumpers Program

```
  10 REM **** spectrum bumpers ****
  15 CLEAR 32499: LET st=32500
  20 GO SUB 500: REM initialise
  25 GO SUB 3000: REM load machine code
  30 LET dr=fw
  40 REM **** pulse forwards ****
  50 LET cc=0
  60 GO SUB 1000: LET cc=cc+1: REM pulse
  70 REM ** test bumpers **
  80 LET nm=192: GO SUB 2000: REM perform AND
  90 IF USR st=192 THEN  GO TO 60
 100 REM **** back to start ****
 110 LET dr=bw
 120 FOR i=1 TO cc
 130 GO SUB 1000: REM pulse
 140 NEXT i
 150 OUT 31,0: STOP
 500 REM **** initialise ****
 510 LET fw=4: LET bw=2: LET lf=6: LET rt=0
 520 RETURN
1000 REM **** pulse ****
1010 OUT 31,dr+9
1020 OUT 31,dr+1
1030 RETURN
2000 REM **** perform AND ****
2010 POKE st+1,IN 31
2020 POKE st+3,nm: RETURN
3000 REM **** machine code loader ****
3010 FOR i=st TO st+8
3020 READ a: POKE i,a
3030 NEXT i
3040 DATA 62,0,14,0,161,6,0,79,201
3050 RETURN
```

Basic Flavours

In the Linear Calibration program on page 894, delete lines 20, 30 and 70 and make the following changes:

```
 80 LET dr=fw
100 LET a$=INKEY$: IF a$="" THEN GO TO 100
280 OUT 31,dr+9
290 OUT 31,dr+1
```

In the Angular Calibration program on page 895, delete lines 20, 30 and 50 and make the following changes:

```
 60 LET dr=rt
 80 LET dr=fw
100 OUT 31,dr+9
110 OUT 31,dr+1
```

In the Robot Measure program on page 945, delete lines 1010, 1020, 1030, 7010 and 7510 and make the following changes:

```
2050 GO SUB 8100: IF USR st<>lb THEN GO TO
     2040
2090 GO SUB 8100: IF USR st<>lb THEN GO TO
     2080
2150 GO SUB 8100: IF USR st<>rb THEN GO TO
     2140
2190 GO SUB 8100: IF USR st<>rb THEN GO TO
     2180
2200 OUT 31,0
3010 CLS
3520 GO SUB 8100: IF USR st=nb THEN GO TO 3510
4010 GO SUB 8100: IF USR st=rb THEN LET ss=rt:
     GO SUB 5000: RETURN
4020 GO SUB 8100: IF USR st=lb THEN LET ss=lf:
     GO SUB 5000: RETURN
5020 GO SUB 8100: IF USR st=nb THEN GO TO 5010
6080 GO SUB 8100: IF USR st=nb THEN GO TO 6070
8100 REM **** perform AND ****
8110 POKE st+1, 192: POKE st+3, IN 31: RETURN
8200 REM **** machine code loader ****
8210 FOR i=st TO st+8
8220 READ a: POKE i, a
8230 NEXT i
8240 DATA 62,0,14,0,161,6,0,79,201
8250 RETURN
```

THE OLD AND THE NEW

We begin a two-part investigation of how BASIC interacts with the operating system of the BBC Micro. Here, we look at the two versions of BASIC used on the computer, consider how the 'language workspace' is arranged, and discuss BBC BASIC's error handling facilities.

The BASIC ROM in the BBC Micro occupies the memory space between addresses &8000 and &BFFF (see the memory map on page 879). This area of memory is also known as the paged ROM space. This means that, by a judicious use of paged addressing, several different ROM chips can be co-resident in the machine and use the same block of memory. Only one of these ROMs is active, or 'paged in', at any one time. Examples of ROMs that occupy this area of memory are the DFS chip, Econet chip, utilities such as Wordwise, View or Disk Doctor, and other languages, such as FORTH and BCPL. The advantage of using paged ROMs is quite clear: the technique saves memory.

Exactly which paged ROM is active at any time depends upon the value held in the *paging register*, which is part of the hardware controlled by the OS. It is capable of accessing up to 16 paged ROMs at any time. Paged ROMs are examined by the OS when something happens that it isn't expecting. For example, if a * command is issued that the OS doesn't recognise, it interrogates any paged ROMs present in the machine before issuing the BAD COMMAND error message. *INFO, for example, would not be picked up by the OS ROM and so would be passed over to any paged ROMs in the machine. If fitted, the DFS ROM would accept this command and act on it.

VERSIONS OF BBC BASIC

Two versions of BASIC have been designed for BBC Micros in use in the UK. These are BASIC I and BASIC II. To see which is resident in your machine, press CTRL-Break and type in REPORT, then press Return. If a copyright message containing the date 1982 is printed to the screen, you have BASIC II. If 1981 is returned, you have the older BASIC I dialect. The later version cleared up a few bugs that were lurking in BASIC I and added more features.

These new features include a filing command, OPENUP, and a change in the role played by OPENIN. In BASIC II, OPENIN will open up a file for INPUT only; in BASIC I a file could be opened for both input and output using this command — a valuable feature for random access files on disk. OPENUP in BASIC II opens a file for both input and output, and so can be seen as the equivalent of the BASIC I OPENIN. This particular alteration has caused trouble when programs are transferred between machines. Should you write a program involving OPENIN on BASIC II, and then reload the program on a machine with BASIC I fitted, you've got problems! The token used to represent OPENIN in BASIC II is not recognised by BASIC I. Thus, the following statement:

```
Y%=OPENIN("FRED")
```

on BASIC II is read as:

```
Y%=("FRED")
```

on BASIC I, and this gives an error message.

The other new features in BASIC II are of particular interest to machine code programmers, and one major improvement deserves special mention. In BASIC I, if it was necessary to incorporate data into machine code programs, you had to leave the BBC assembler and use the ?, ! and $ operators to put the data into memory. For example:

```
1000 LDA data
1010 .data
1020 ]
1030 ?P%=00
1040 P%=P%+1
1050 OPT pass
```

This is a little longwinded, and most assemblers offer us a way around this by using *assembler directives*. These are instructions that tell the assembler to do a particular job — in our program, OPT is an assembler directive. BASIC II, however, provides us with four more assembler directives: EQUB reserves a single byte of memory; EQUD reserves two bytes; EQUW four bytes; and EQUS allows us to store a string in memory. Our BASIC I program, therefore, could be rewritten in BASIC II as:

```
1000 LDA data
1010 .data EQUB 00
1020 . . . rest of program
```

This would result in the A register being loaded with the byte held in address data, which is zero in this example. A further example is:

```
1000 .message EQUS "Hi There!"
```

In this case, the bytes representing the message Hi There! would be placed in memory at the address labelled .message. When using all of these assembler directives, P% is automatically updated to take the data into account.

Another useful addition to BASIC II is the

command OSCLI. We have already discussed how to pass * commands to the BBC OS using the OSCLI call at address &FFF7 (see page 897). Using BASIC I, we had to set up the X% and Y% variables — or the X and Y registers — with the address in memory of the string that represented the command to be passed. The new OSCLI command in BASIC II makes this task much easier. Compare the two versions:

BASIC I	BASIC II
$&A00="*TAPE" X%=&A00 MOD 256 Y%=&A00 DIV 256 CALL &FFF7	OSCLI("*TAPE")

As well as these two versions of BASIC, there is a modified BASIC II for the US market; the alterations made in this version are simply to take different display characteristics into account. Also, the 6502 second processor has its own version of BASIC called HI BASIC. This appears to be a mixture of the US version of BASIC II and BASIC II itself. To all extents and purposes it is identical to BASIC II from the programmer's point of view. Finally, we should add that both versions of BASIC effectively use the memory of the BBC Micro in the same way.

LANGUAGE WORKSPACE

The area of memory between &400 and &7FF is known as the *language workspace;* this is reserved for the language currently being used. In addition, the zero page between addresses &00 and &6F is reserved for the language in current use, although some of the space between &50 and &6F doesn't appear to be used very much. Page 4 of the memory is used by BASIC for the storage of variables; &400 to &46B is the area of memory where the values assigned to the resident integer variables, such as A% or X%, are stored. @% is stored in the four bytes from &400 to &403 (with the least significant byte of its value in &400), A% is stored from &404 to &407 — and so on right up to the last resident integer variable, Z%.

The area of memory between &480 and &4F9 is used as an index to the BASIC interpreter; it points to the places in memory where the other floating point or integer variables, arrays or string variables are stored. For example, the two bytes in &482 and &483 point to an area of memory — between TOP and HIMEM — where the details of all the currently used variables with names beginning with the letter A can be found. The address held in the locations &4F6 and &4F7 refers to the storage of BASIC procedures, and the address stored in &4F8 and &4F9 refers to the storage of functions.

The area of memory between &500 and &5FF is used by the BASIC interpreter as a stack. This should not be confused with the 6502 stack that is stored in page 1 of the memory. Page 5 is used by BASIC to store the RETURN details for any GOSUB calls, and the correct information needed for the proper execution of FOR . . . NEXT loops and REPEAT . . . UNTIL loops.

Page 6 of the memory is used by BASIC either when it's handling string variables or when the extended version of the CALL command is used to execute a machine code program (we'll look at this in more detail later in the course). Page 7 is used by BBC BASIC as an input buffer when we're typing in BASIC commands or program lines. Whatever we type in is placed in this area of memory, awaiting processing. If it's a direct command, it is then interpreted and executed; if a program line, it's placed in the program at the appropriate position. That concludes our brief detailing of the use of the BBC Micro's language workspace by BASIC. The useful thing about this area of memory is that if we're not using BASIC at all, it is available for us to use otherwise.

BASIC ERROR HANDLING

We've all made errors in our BASIC programs, and have had the computer bite back with its error messages. What's rather interesting about the way in which errors are generated by BASIC on the BBC Micro is that we can make use of the 'error handler' to produce our own error statements, complete with error numbers and messages. This is most useful if we've got some machine code programs that are to be used in conjunction with BASIC. The routine that handles the errors generated by a BASIC program is pointed to by the content of a vector called BRKV, which is located at address &202 and &203. Let's look at how the routine pointed to by this vector is entered.

When BASIC wants to generate an error because of some illegal event that has just occurred — a 'division by zero', for example — it executes a 6502 machine code instruction called BRK. This causes the 6502 to jump to the routine specified by the address held in BRKV. Under normal circumstances, this routine expects to find the BRK instruction followed by a series of bytes, set up in this format:

BRK
Error Number (one byte)
Error Message (a series of bytes)
Message End (a zero byte)

Thus, for error number 4, which generates the 'Mistake' message, the following routine would be executed to cause entry to the error handler.

Byte	Comment
BRK	Instruction
4	Error number
M	
i	
s	ASCII codes
t	of error
a	message
k	
e	
00	Message end

The BASIC error handler then sets the system variable ERL to hold the line number where the

error occurred. ERR is then set up with the error number — in this case 4 — and the handler ensures that the next time REPORT is evaluated, the error message 'Mistake' will be printed.

The BRK instruction can be used in your own machine code programs, and therefore you can define your own error messages. Any error message thus generated will be picked up by the BASIC ON ERROR GOTO instruction, as if the error had been generated by the BASIC interpreter. For example, the following short section of a machine code program returns a 'Number too big' error message:

```
1010 BRK
1020 EQUB 254
1030 EQUS "Number too big"
1040 EQUB 00
```

Once executed, printing ERR will return 254 and REPORT will give 'Number too big'.

It's possible to alter the contents of this vector and change the way in which the BBC Micro responds to errors. By using this vector you can add new commands to the BBC BASIC, but this requires some rather sophisticated programming.

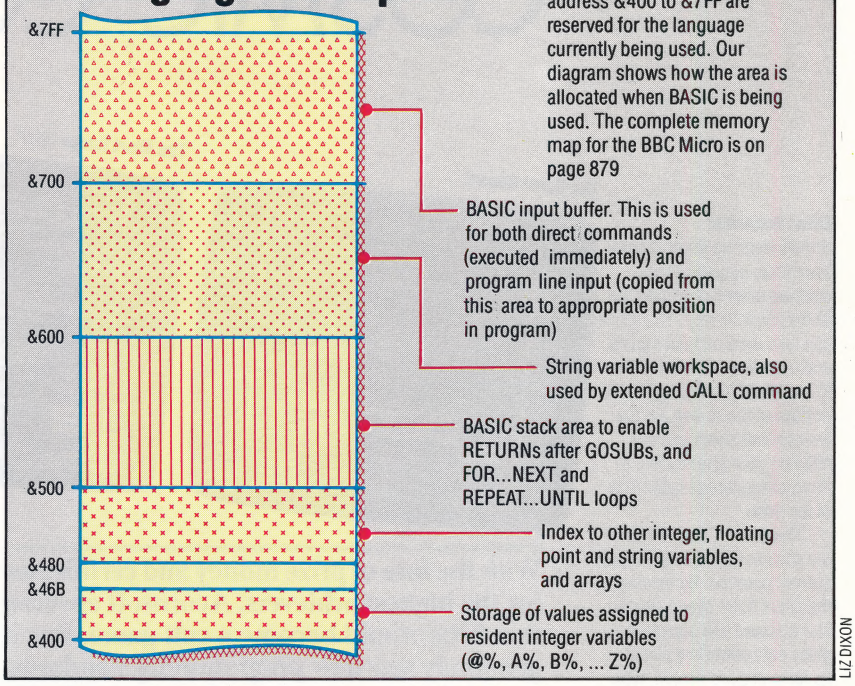
In the next instalment of the course, we'll look at the way in which BASIC interacts with the BBC Micro's operating system.

Routine Examinations

We list here the addresses in ROM of some of the BBC Micro's BASIC routines mentioned so far in the series. By examining these routines you may gain further insights into how BASIC operates. Not all the routines listed here terminate with an RTS instruction, so it is not recommended that you call them indiscriminately from your own programs, but examination of the techniques used could help you to improve your own programming skills

ROM Entry Point	&8000
BGET statement	&BF6F
BPUT statement	&BF58
CHAIN statement	&BF2A
CLOSE statement	&BF99
EOF function	&ACB8
INKEY function	&ACAD
INKEY$ function	&BO26
INPUT statement	&BA44
INPUT# statement	&B9CF
LOAD command	&BF24
OPENIN function	&BF78
OPENOUT function	&BF7C
OPENUP function	&BF80
PRINT# statement	&8D2B
SAVE command	&BEF3
TIME statement	&92C9
TIME function	&AEB4
VDU statement	&942F

The Language Workspace



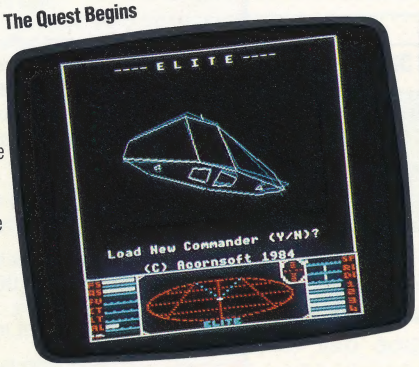
LIZ DIXON

Variations Between BBC BASIC I And BASIC II

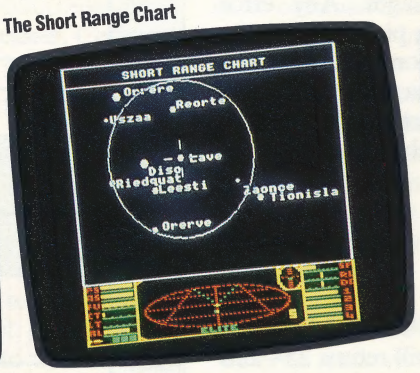
Command Or Bug	BASIC I	BASIC II
REPORT	Gives 1981 message	Gives 1982 message
OPENUP	Not implemented	Opens a file for reading and writing (same as BASIC I OPENIN command)
OPENIN	Opens a file for read and write	Opens a file for read only
OPT n	OPT 1 to 3 are implemented only	OPT 1 to 7 are now implemented. The four new commands (OPTs 4 to 7) are the same as the earlier three, but the code is assembled to the address in the O% variable
O%	No special use	Given function outlined in previous entry (OPT n commands)
EQUB, EQUD, EQUS, EQUW	Not implemented	Allows the programmer to include data in machine code programs without leaving the assembler (see main text)
INSTR (a$,b$)	This was a serious bug: if a$ was shorter than b$, then a crash was possible	Debugged
Real Nos.	Stored to nine-figure accuracy	Stored to 10-figure accuracy
OSCLI	Not implemented	Allows easy passing of commands to OS from BASIC string variables

COSMIC COMMERCE

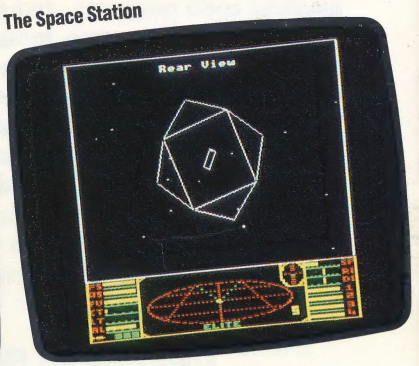
The Quest Begins



The Short Range Chart



The Space Station



Elitist Behaviour
Shown here are three scenes from Elite, including the opening screen depicting the Cobra Mark III ship. The game can take months to complete and therefore has the facility to be SAVEd in midplay. At the bottom of the screen are the various indicators you will need on your journey and a three-dimensional radar map of the area.

The second screen shows the stars in the vicinity of your current position. By moving the cross hairs over a chosen star a screen will appear that gives data about the planet — its form of government, type of economy, etc. The third screen shows the space station. To dock with the station the player has to guide his spaceship into a small rectangular port

With the lure of prize money and certificates for the highest achievers, Acornsoft ensured that Elite would become a cult game. But it genuinely sets new programming standards, combining the elements of arcade action in 3D, adventure and strategy into an obsessively addictive challenge.

Acornsoft's Elite combines elements of arcade, adventure and strategy. The aim of the game is to become one of the Elite. To achieve this the player must become a successful trader among the thousands of planets throughout eight galaxies.

To stay alive long enough to make his fortune and join the Elite, the player not only has to become a skilled navigator and fighter, he also needs an eye for good business, a certain amount of low cunning and to be able to tread the thin line between what is legal and what is illegal. Large profits are to be made quickly by trading contraband between planets or by becoming a pirate, but the player has to be wary. The police are likely to be notified and he may soon find himself labelled an offender or a fugitive — and the fast police craft shoots to kill.

The game begins with the player docked in a space station above the planet Lave. Your craft is a Cobra Mark III; it is equipped with a front pulse laser and has a cargo capacity of 20 tonnes. Before setting out you are advised to become familiar with the many controls and options available and to practice docking manoeuvres, otherwise you could crash on your first mission. If you spend too much time trying to dock with a space station circling a hostile planet you will surely attract the attention of the pirates.

Once docked, it's time to do business. By comparing the prices available on the planet with the goods in your hold, you must decide whether to sell or hold on until you find a more favourable rate elsewhere. One of the few drawbacks of the game, which it has in common with most other trading games available for computers, is that there is no option for bargaining. The player either takes the price offered or leaves it.

The real art of trading lies in deciding which products to take to which planet. For example, food can be bought cheaply on an agricultural planet and sold at a good price on an industrial planet. Machinery can then be taken back to the agricultural planet. The player must also take into account the political structure of the planet and the type of beings that inhabit it. A disorganised political structure means that pirates are likely to be around, and some species of alien are more likely to swindle or kill you than others.

The three-dimensional graphics of Elite are excellent. The planets, space stations and spaceships are drawn in diagrammatic form, which at first takes a little getting used to, but one soon begins to appreciate the advantage of higher resolution and the ability to rotate in three dimensions. This is used to greatest effect in docking and in combat. Space stations have only a single entrance that must be approached at the right angle. When you are engaged in combat, the ability to see the enemy craft (of which there are many different types) adds greatly to the realism of the game.

In addition to the cassette or disk, you receive a comprehensive manual detailing every aspect of the game, a story based on the game called *The Dark Wheel*, a card containing a summary of the controls, a template for the function keys and a poster showing the various types of spacecraft one is likely to encounter. It is not surprising that Elite has become the best selling game for the BBC Micro.

Elite: For the BBC Micro and the Electron (joysticks optional)
Prices: BBC cassette version: £14.95; disk version: £17.95; Electron cassette: £12.95
Publishers: Acornsoft Ltd, Betjemin House, 104 Hills Road, Cambridge, CB2 1LQ
Authors: Ian Bell, David Braben
Format: Cassette or disk

HAVE YOU ORDERED YOUR VOLUME FIVE BINDER YET?

FOLD 2

IF YOU HAVE NOT ASKED FOR THE BINDERS TO BE SENT TO YOU AUTOMATICALLY, DO SO NOW, AND ENSURE THAT YOUR COPIES OF THE HOME COMPUTER ADVANCED COURSE ARE KEPT PROPERLY BOUND AND IN GOOD CONDITION FOR YEARS TO COME.

BY TICKING THE BOX OPPOSITE, YOU WILL BE SENT BINDER NUMBER 5.

☐ PLEASE SEND ME MY VOLUME 5 BINDER NOW. I ENCLOSE A CHEQUE/POSTAL ORDER FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING).

FOLD 3

BY TICKING THE OTHER BOX *AS WELL*, YOU WILL BE SENT SUBSEQUENT BINDERS FOR YOUR COLLECTION.

☐ I WOULD ALSO LIKE TO RECEIVE FUTURE BINDERS AS THEY ARE ISSUED. I UNDERSTAND THAT I WILL RECEIVE A PAYMENT ADVICE FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING) WITH EACH BINDER. IF I AM NOT SATISFIED WITH THE BINDER, I CAN RETURN IT TO YOU, WITHIN 14 DAYS, AND OWE NOTHING.

NO STAMP NECESSARY.

JUST FOLD UP THE PAGE AS INDICATED, REMEMBERING TO ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING, AND SEND TO US TODAY.

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

		INITIALS	SURNAME
MR			
MRS			
MISS			

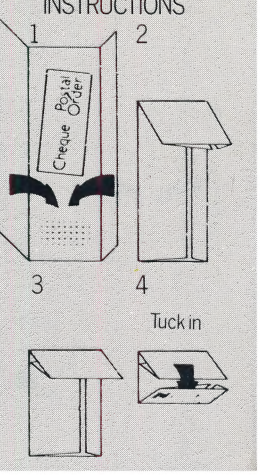
ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

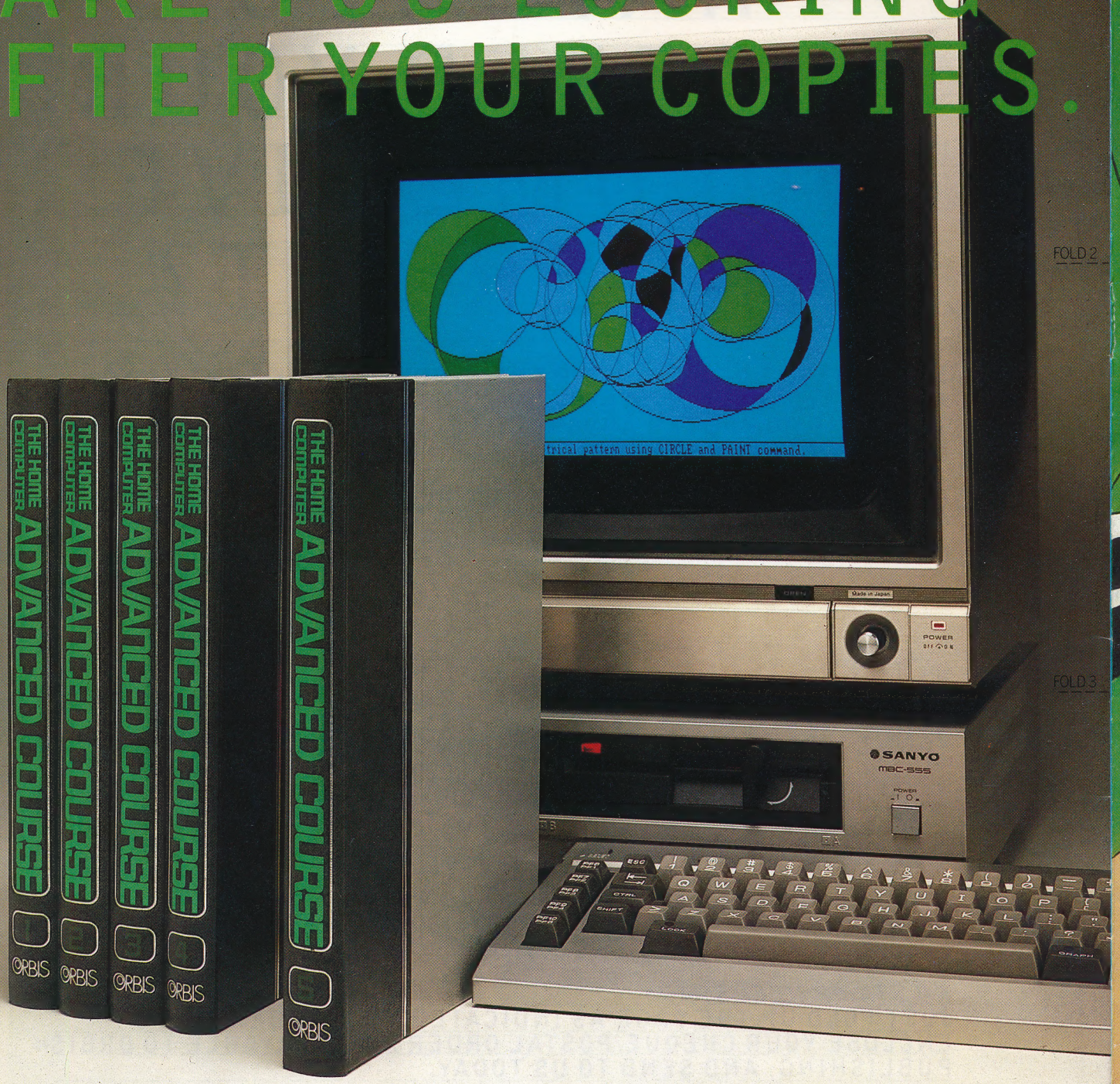
FOLD 1

FOLD 1

FOLD 4
FOLDING INSTRUCTIONS



ARE YOU LOOKING AFTER YOUR COPIES.

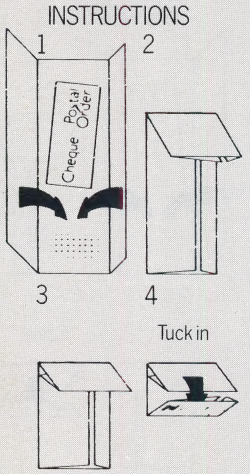


FOLD 2

FOLD 3

FOLD 4

FOLD 4
FOLDING INSTRUCTIONS



FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

FOLD 1

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE

GUARANTEE

If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.

GUARANTEE